

Nesting Depth of Operators in Graph Database Queries: Expressiveness Vs. Evaluation Complexity

M. Praveen and B. Srivathsan*
Chennai Mathematical Institute, India

Abstract

Designing query languages for graph structured data is an active field of research, where expressiveness and efficient algorithms for query evaluation are conflicting goals. To better handle dynamically changing data, recent work has been done on designing query languages that can compare values stored in the graph database, without hard coding the values in the query. The main idea is to allow variables in the query and bind the variables to values when evaluating the query. For query languages that bind variables only once, query evaluation is usually NP-complete. There are query languages that allow binding inside the scope of Kleene star operators, which can themselves be in the scope of bindings and so on. Uncontrolled nesting of binding and iteration within one another results in query evaluation being PSPACE-complete.

We define a way to syntactically control the nesting depth of iterated bindings, and study how this affects expressiveness and efficiency of query evaluation. The result is an infinite, syntactically defined hierarchy of expressions. We prove that the corresponding language hierarchy is strict. Given an expression in the hierarchy, we prove that it is undecidable to check if there is a language equivalent expression at lower levels. We prove that evaluating a query based on an expression at level i can be done in Σ_i in the polynomial time hierarchy. Satisfiability of quantified Boolean formulas can be reduced to query evaluation; we study the relationship between alternations in Boolean quantifiers and the depth of nesting of iterated bindings.

1 Introduction

Graph structures representing data have found many applications like semantic web [11], social networks [19] and biological networks [13]. Theoretical models of such data typically have a graph with nodes representing entities and edges representing relations among them. One reason for the popularity of these models is their flexibility in handling semi-structured data. While traditional relational databases impose rigid structures on the relations between data elements, graph databases are better equipped to handle data in which relations are not precisely known and/or developing dynamically.

A fundamental query language for such models is Regular Path Queries (RPQs), which is now part of the W3C recommendation [18]. An RPQ consists of a regular expression over the finite alphabet labeling the edges of the graph. Suppose a communication network is modeled by a graph, where nodes represent servers and edges labeled ℓ represent links between them. Evaluating the RPQ ℓ^* on this graph results in the set of pairs of nodes between which there exists a route. Suppose each link has a priority and we need pairs of connected nodes where all intermediate links have the same priority. We can hard code the set of priorities in the query. If the set of priorities is not static, a querying mechanism which avoids hard coding is better. Every edge can be labeled by a supplementary data value (priority of the link, in this example) and we want query languages that can compare data values without hard coding them in the syntax. Nodes can also carry data values. In generic frameworks, there is no a priori bound on the number of possible data values and they are considered to be elements of an infinite domain. Graph databases with data values are often called data graphs in theory and property graphs in practice.

*Both the authors are partially funded by a grant from Infosys Foundation.

One way to design querying languages for data graphs is to extend RPQs using frameworks that handle words on infinite alphabets [16, 15, 12, 23]. Expressiveness and efficient algorithms for query evaluation are conflicting goals for designing such languages. We study a feature common to many of these languages, and quantify how it affects the trade-off between expressiveness and complexity of query evaluation. Variable finite automata [10] and parameterized regular expressions [2] are conservative extensions of classical automata and regular expressions. They have variables, which can be bound to letters of the alphabet at the beginning of query evaluation. The query evaluation problem is NP-complete for these languages. Regular expressions with binding (REWBs) [15] is an extended formalism where binding of variables to values can happen inside a Kleene star, which can itself be in the scope of another binding operator and so on. Allowing binding and iteration to occur inside each other’s scope freely results in the query evaluation problem being PSPACE-complete. Here we study how the expressiveness and complexity of query evaluation vary when we syntactically control the depth of nesting of iterated bindings.

Contributions:

1. We syntactically classify REWBs according to the depth of nesting of iterated bindings.
2. The resulting hierarchy of data languages is strict, and so is the expressiveness of queries.
3. It is undecidable to check if a given REWB has a language equivalent one at lower levels.
4. An REWB query in level i can be evaluated in Σ_i in the polynomial time hierarchy.
5. For lower bounds, we consider quantified Boolean formulas with some restrictions on quantifications and reduce their satisfiability to query evaluation, with some restrictions on the queries.

For proving strictness of the language hierarchy, we build upon ideas from the classic star height hierarchy [9]. Universality of REWBs is known to be undecidable [17, 12]. We combine techniques from this proof with tools developed for the language hierarchy to prove the third result above. The Σ_i upper bound for query evaluation involves complexity theoretic arguments based on the same tools. In the reductions from satisfiability of quantified Boolean formulas to the query evaluation problem, the relation between the number of alternations (in the Boolean quantifiers) and the depth of nesting (of iterated bindings in REWBs) is not straight forward. We examine this relation closely in the framework of parameterized complexity theory, which is suitable for studying the effect of varying the structure of input instances on the complexity.

Related work: The quest for efficient evaluation algorithms and expressive languages to query graph databases, including those with data values, is an active area of research; [1] is a recent comprehensive survey. Numerous formalisms based on logics and automata exist for handling languages over infinite alphabets [20]. In [16], the suitability of these formalisms as query languages has been studied, zeroing in on register automata mainly for reasons of efficient evaluation. The same paper introduced regular expressions with memory and proved that they are equivalent to register automata. REWBs [15] have slightly less expressive power but have better scoping structure for the binding operator. Properties of these expressions have been further studied in [12]. In [14], XPath has been adapted to query data graphs. Pebble automata have been adapted to work with infinite alphabets in [17]. A strict language hierarchy based on the number of pebbles allowed in pebble automata has been developed in [22]. Many questions about comparative expressiveness of register and pebble automata are open [17]. Fixed-point logics can be used to define languages over infinite alphabets [4]. These logics can use the class successor relation, which relates two positions with the same data value if no intermediate position carries the same value. Expressiveness of these logics increase [6, 5], when the number of alternations between standard successor relation and class successor relation increase.

2 Preliminaries

2.1 Data Languages and Querying Data Graphs

We follow the notation of [15]. Let Σ be a finite alphabet and \mathcal{D} a countably infinite set. The elements of \mathcal{D} are called *data values*. A *data word* is a finite string over the alphabet $\Sigma \times \mathcal{D}$. We will write a data word as

$(\overset{a_1}{d_1})(\overset{a_2}{d_2}) \dots (\overset{a_n}{d_n})$, where each $a_i \in \Sigma$ and $d_i \in \mathcal{D}$. A set of data words is called a *data language*.

An extension of standard regular expressions, called *regular expressions with binding (REWB)*, has been defined in [15]. Here, data values are compared using variables. For a set $\{x_1, x_2, \dots, x_k\}$ of variables, the set of conditions \mathcal{C}_k is the set of Boolean combinations of x_i^- and x_i^+ for $i \in \{1, \dots, k\}$. A data value $d \in \mathcal{D}$ and a partial valuation $\nu : \{x_1, \dots, x_k\} \rightarrow \mathcal{D}$ satisfies the condition x_i^- (written as $d, \nu \models x_i^-$) if $\nu(x_i) = d$. The satisfaction for other Boolean operators are standard.

Definition 2.1 (Regular expressions with binding (REWB) [15]). *Let Σ be a finite alphabet and $\{x_1, \dots, x_k\}$ a set of variables. Regular expressions with binding over $\Sigma[x_1, \dots, x_k]$ are defined inductively as: $r := \varepsilon \mid a \mid a[c] \mid r + r \mid r \cdot r \mid r^* \mid a \downarrow_x (r)$ where $a \in \Sigma$ is a letter in the alphabet, $c \in \mathcal{C}_k$ is a condition on the variables and $x \in \{x_1, \dots, x_k\}$ is a variable.*

We call \downarrow_x the *binding operator*. In the expression $a \downarrow_x (r)$, the expression r is said to be the *scope* of the binding \downarrow_x . A variable x in an expression is *bound* if it occurs in the scope of a binding \downarrow_x . Otherwise it is *free*. We write $fv(r)$ to denote the set of free variables in r and $r(\bar{x})$ to denote that \bar{x} is the sequence of all free variables. The semantics of an REWB $r(\bar{x})$ over the variables $\{x_1, \dots, x_k\}$ is defined with respect to a partial valuation $\nu : \{x_1, \dots, x_k\} \rightarrow \mathcal{D}$ of the variables. A valuation ν is *compatible* with $r(\bar{x})$ if $\nu(\bar{x})$ is defined.

Definition 2.2 (Semantics of REWB). *Let $r(\bar{x})$ be an REWB over $\Sigma[x_1, \dots, x_k]$ and let $\nu : \{x_1, \dots, x_k\} \rightarrow \mathcal{D}$ be a valuation of variables compatible with $r(\bar{x})$. The language of data words $L(r, \nu)$ defined by $r(\bar{x})$ with respect to ν is given as follows:*

r	$L(r, \nu)$	r	$L(r, \nu)$	r	$L(r, \nu)$
ε	$\{\varepsilon\}$	a	$\{(\overset{a}{d}) \mid d \in \mathcal{D}\}$	$a[c]$	$\{(\overset{a}{d}) \mid d, \nu \models c\}$
$r_1 + r_2$	$L(r_1, \nu) \cup L(r_2, \nu)$	$r_1 \cdot r_2$	$L(r_1, \nu) \cdot L(r_2, \nu)$	r_1^*	$(L(r_1, \nu))^*$
$a \downarrow_{x_i} (r_1)$	$\bigcup_{d \in \mathcal{D}} \{(\overset{a}{d})\} \cdot L(r_1, \nu[x_i \rightarrow d])$				

where $\nu[x_i \rightarrow d]$ denotes the valuation which is the same as ν except for x_i which is mapped to d . An REWB r defines the data language $L(r) = \bigcup_{\nu \text{ compatible with } r} L(r, \nu)$.

For example, the REWB $a \downarrow_x (b[x^-]^*)$ defines the set of data words of the form ab^* with all positions having the same data value. The REWB $(a \downarrow_x (b[x^-]))^*$ defines the set of data words of the form $(\overset{a}{d_1})(\overset{b}{d_1})(\overset{a}{d_2})(\overset{b}{d_2}) \dots (\overset{a}{d_n})(\overset{b}{d_n})$.

Definition 2.3 (Data graphs). *A data graph G over a finite alphabet Σ and an infinite set of data values \mathcal{D} is a pair (V, E) where V is a finite set of vertices, and $E \subseteq V \times \Sigma \times \mathcal{D} \times V$ is a set of edges which carry labels from $\Sigma \times \mathcal{D}$.*

We do not have data values on vertices, but they can be introduced without affecting the results. A *regular data path query* is of the form $Q = x \xrightarrow{r} y$ where r is an REWB. Evaluating Q on a data graph G results in the set $Q(G)$ of pairs of nodes $\langle u, v \rangle$ such that there exists a data path from u to v and the sequence of labels along the data path forms a data word in $L(r)$. Evaluating a regular data path query on a data graph is known to be PSPACE-complete in general and NLOGSPACE-complete when the query is of constant size [15]. We sometimes identify the query Q with the expression r and write $r(G)$ for $Q(G)$. A query r_1 is said to be contained in another query r_2 if for every data graph G , $r_1(G) \subseteq r_2(G)$. It is known from [12, Proposition 3.5] that a query r_1 is contained in the query r_2 iff $L(r_1) \subseteq L(r_2)$. Hence, if a class E_2 of REWBs is more expressive than the class E_1 in terms of defining data languages, E_2 can also express more queries than E_1 .

2.2 Parameterized Complexity

The size of queries are typically small compared to the size of databases. To analyze the efficiency of query evaluation algorithms, the size of the input can be naturally split into the size of the query and the size of the database. Parameterized complexity theory is a formal framework for dealing with such problems. An instance of

a parameterized problem is a pair (x, k) , where x is an encoding of the input structure on which the problem has to be solved (e.g., a data graph and a query), and k is a parameter associated with the input (e.g., the size of the query). A parameterized problem is said to be in the parameterized complexity class Fixed Parameter Tractable (FPT) if there is a computable function $f : \mathcal{N} \rightarrow \mathcal{N}$, a constant $c \in \mathcal{N}$ and an algorithm to solve the problem in time $f(k)|x|^c$.

We will see later that the query evaluation problem is unlikely to be in FPT, when parameterized by the size of the regular data path query. There are many parameterized complexity classes that are unlikely to be in FPT, like $W[\text{SAT}]$, $W[P]$, $AW[\text{SAT}]$ and $AW[P]$. To place parameterized problems in these classes, we use FPT-reductions.

Definition 2.4 (FPT reductions). *A FPT reduction from a parameterized problem Q to another parameterized problem Q' is a mapping R such that:*

1. *For all instances (x, k) of parameterized problems, $(x, k) \in Q$ iff $R(x, k) \in Q'$.*
2. *There exists a computable function $g : \mathcal{N} \rightarrow \mathcal{N}$ such that for all (x, k) , say with $R(x, k) = (x', k')$, we have $k' \leq g(k)$.*
3. *There exist a computable function $f : \mathcal{N} \rightarrow \mathcal{N}$ and a constant $c \in \mathcal{N}$ such that R is computable in time $f(k)|x|^c$.*

3 Nesting Depth of Iterated Bindings and Expressive Power

A binding \downarrow_x along with a condition $[x=]$ or $[x\neq]$ is used to constrain the possible data values that can occur at certain positions in a data word. A binding inside a star — an *iterated binding* — imposes the constraint arbitrarily many times. For instance, the expression $r_1 := (a_1 \downarrow_{x_1} (b_1[x_1^-]))^*$ defines data words in $(a_1 b_1)^*$ where every a_1 has the same data value as the next b_1 . We now define a syntactic mechanism for controlling the nesting depth of iterated bindings. The restrictions result in an infinite hierarchy of expressions. The expressions at level i are generated by F_i in the grammar below, defined by induction on i .

$$\begin{aligned} F_0 &::= \varepsilon \mid a \mid a[c] \mid F_0 + F_0 \mid F_0 \cdot F_0 \mid F_0^* \\ E_i &::= F_{i-1} \mid E_i + E_i \mid E_i \cdot E_i \mid a \downarrow_{x_j} (E_i) \\ F_i &::= E_i \mid F_i + F_i \mid F_i \cdot F_i \mid F_i^* \end{aligned}$$

where $i \geq 1$, $a \in \Sigma$, c is a condition in \mathcal{C}_k and $x_j \in \{x_1, \dots, x_k\}$. Intuitively, E_i can add bindings over iterations (occurring in F_{i-1}) and F_i can add iterations over bindings (occurring in E_i). The nesting depth of iterated bindings in an expression in F_i is therefore i . The union of all expressions at all levels equals the set of REWBs. In this paper, we use subscripts to denote the levels of expressions and superscripts to denote different expressions in a level: so e_5^1 is some expression in E_5 , f_3^2 is some expression in F_3 .

We now give a sequence of expressions $\{r_i\}_{i \geq 1}$ such that each r_i is in F_i but no language equivalent expression exists in F_{i-1} . For technical convenience, we use an unbounded number of letters from the finite alphabet and an unbounded set of variables. The results can be obtained with a constant number of letters and variables.

Definition 3.1. *Let $\{a_1, b_1, a_2, b_2, \dots\}$ be an alphabet and $\{x_1, x_2, \dots\}$ a set of variables. We define r_1 to be $(a_1 \downarrow_{x_1} (b_1[x_1^-]))^*$. For $i \geq 2$, define $r_i := (a_i \downarrow_{x_i} (r_{i-1} b_i[x_i^-]))^*$.*

From the syntax, it can be seen that each r_i is in F_i . To show that $L(r_i)$ cannot be defined by any expression in F_{i-1} , we will use an “automaton view” of the expression, as this makes pigeon-hole arguments simpler. No automata characterizations are known for REWBs in general; the restrictions on the binding and star operators in the expressions of a given level help us build specific automata in stages.

Standard finite state automata can be converted to regular expressions by considering generalized non-deterministic finite automata, where transitions are labeled with regular expressions instead of a single letter (see e.g., [21, Lemma

1.32]). The language of an expression f_i^1 can be accepted by such an automaton, where transitions are labeled with expressions in E_i . We will denote this automaton by $\mathcal{A}(f_i^1)$. Similarly, the language of an expression e_i^1 can be accepted by an automaton whose transitions are labeled with expressions in F_{i-1} or with $a \downarrow_x$. We will denote this automaton by $\mathcal{A}(e_i^1)$. There are no cycles in $\mathcal{A}(e_i^1)$, since e_i^1 can not use the Kleene $*$ operator except inside expressions in F_{i-1} . The runs of $\mathcal{A}(e_i^1)$ are sequences of pairs of a state and a valuation for variables. The valuations are updated after every transition with a label of the form $a \downarrow_x$. Formal semantics are given in Appendix A, which also contains all the proofs in detail.

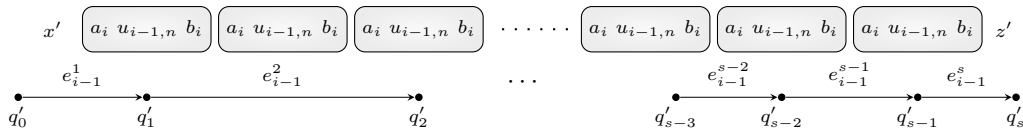
We will prove that $L(r_i)$ cannot be defined by any expression in E_i (and hence not by any expression in F_{i-1}). We first define the following sequence of words, which will be used in the proof. Let $\{d[j_1, j_2] \in \mathcal{D} \mid j_1, j_2 \in \mathcal{N}\}$ be a set of data values such that $d[j_1, j_2] \neq d[j'_1, j'_2]$ if $\langle j_1, j_2 \rangle \neq \langle j'_1, j'_2 \rangle$. For every $n \geq 1$, define the words:

$$\begin{aligned} u_{1,n} &:= \begin{pmatrix} a_1 \\ d[1, 1] \end{pmatrix} \begin{pmatrix} b_1 \\ d[1, 1] \end{pmatrix} \begin{pmatrix} a_1 \\ d[1, 2] \end{pmatrix} \begin{pmatrix} b_1 \\ d[1, 2] \end{pmatrix} \cdots \begin{pmatrix} a_1 \\ d[1, n^2] \end{pmatrix} \begin{pmatrix} b_1 \\ d[1, n^2] \end{pmatrix} \\ u_{i,n} &:= \begin{pmatrix} a_i \\ d[i, 1] \end{pmatrix} u_{i-1,n} \begin{pmatrix} b_i \\ d[i, 1] \end{pmatrix} \begin{pmatrix} a_i \\ d[i, 2] \end{pmatrix} u_{i-1,n} \begin{pmatrix} b_i \\ d[i, 2] \end{pmatrix} \cdots \begin{pmatrix} a_i \\ d[i, n^2] \end{pmatrix} u_{i-1,n} \begin{pmatrix} b_i \\ d[i, n^2] \end{pmatrix} \\ &\text{for all } i \geq 2 \end{aligned}$$

In order to prove that $L(r_i)$ cannot be defined by any expression in E_i , we will show the following property: if $u_{i,n}$ occurs as a sub-word of a word w in the language of a “sufficiently small” expression e_i^1 , then the same expression accepts a word where some a_j and a matching b_j have different data values. Let $Mismatch_{i,n}$ be the set of all data words obtained from $u_{i,n}$ by modifying the data values so that there exist two positions p, p' with $p < p'$ and a $j \leq i$ such that: p contains $\begin{pmatrix} a_j \\ d \end{pmatrix}$ and p' contains $\begin{pmatrix} b_j \\ d' \end{pmatrix}$ with $d \neq d'$; moreover between positions p and p' , b_j does not occur in the word. We consider expressions in which no two occurrences of the binding operator use the same variable. For an expression e , let $|\mathcal{A}(e)|$ denote the number of states in the automaton $\mathcal{A}(e)$ and $|var(e)|$ denote the number of variables in e .

Lemma 3.2. *Let e_i^1 be an expression and let $n \in \mathcal{N}$ be greater than $(|\mathcal{A}(e)| + 1)$ and $(|var(e)| + 1)$ for every sub-expression e of e_i^1 . Let ν be a valuation of $fv(e_i^1)$ and let x, z be data words. Then: $xu_{i,n}z \in L(e_i^1, \nu) \implies x\bar{u}_{i,n}z \in L(e_i^1, \nu)$ for some $\bar{u}_{i,n} \in Mismatch_{i,n}$.*

Proof idea. By induction on i . Suppose $xu_{i,n}z \in L(e_i^1, \nu)$. The run of $\mathcal{A}(e_i^1)$ on $xu_{i,n}z$ consists of at most n transitions, since the automaton is acyclic and has at most n states. Each of the (at most) n transitions reads some sub-word in the language of some sub-expression f_{i-1}^1 , while the whole word consists of n^2 occurrences of $a_i u_{i-1,n} b_i$. Hence, at least one sub word consists of n occurrences of $a_i u_{i-1,n} b_i$. A run of $\mathcal{A}(f_{i-1}^1)$ on such a sub-word is shown below.



Every transition of this run reads sub-words in the language of some sub-expression e_{i-1}^j . If some transition of this run reads an entire sub-word $u_{i-1,n}$ (as in transition $q'_1 \rightarrow q'_2$), then we can create a mismatch inside this $u_{i-1,n}$ by induction hypothesis. Otherwise, none of the transitions read an a_i and the corresponding b_i together (as in $q'_{s-2} \rightarrow q'_{s-1}$ in the figure). None of the b_i s is compared with the corresponding a_i , so the data value of one of the b_i s can be changed to create a mismatch. The resulting data word will be accepted provided the change does not result in a violation of some condition. Since the range of the valuation has at most $(n-1)$ distinct values, one of the n b_i s is safe for changing the data value. \square

Theorem 3.3. *For any i , the language $L(r_i)$ cannot be defined by any expression in E_i .*

Proof. Suppose r_i is equivalent to an expression e_i^1 . Pick an n bigger than $|\mathcal{A}(e)|$ and $|fv(e)|$ for every sub-expression e of e_i^1 . The word $u_{i,n}$ belongs to $L(r_i)$ and hence $L(e_i^1)$. By Lemma 3.2, we know that if this is the case, then $\bar{u}_{i,n} \in L(r_i)$ for some word $\bar{u}_{i,n} \in \text{Mismatch}_{i,n}$. But $L(r_i)$ cannot contain words with a mismatch. A contradiction. \square

Given an expression at some level, it is possible that its language is defined by an expression at lower levels. Next we show that it is undecidable to check this.

Theorem 3.4. *Given an expression in F_{i+1} , checking if there exists a language equivalent expression in F_i is undecidable.*

Proof idea. By reduction from Post's Correspondence Problem (PCP). The basic idea is from the proof of undecidability of universality of REWBs and related formalisms [17, 15]. For an instance $\{(u_1, v_1), \dots, (u_n, v_n)\}$ of PCP, a solution (if it exists) can be encoded by a data word of the form $w_1 \# r_i \# w_2$, where w_1 is made up of u_i 's, w_2 is made up of v_i 's and r_i is from Definition 3.1. To ensure that such a data word indeed represents a solution, we need to match up the u_i 's in w_1 with the v_i 's in w_2 , which can be done through matching data values. Consider the language of data words of the form $w'_1 \# r_i \# w'_2$ that are *not* solutions of the given PCP instance. This language can be defined by an expression Δ in E_{i+1} , which compares data values in the left of $\# r_i \#$ with those on the right side, to catch mismatches. We can prove that no equivalent expression exists in lower levels, using techniques used in Lemma 3.2. On the other hand, if the given PCP instance does not have a solution, no data word encodes a solution, so the given language is defined by $\Sigma^* r_i \Sigma^*$, which is in F_i . \square

4 Complexity of Query Evaluation

In this section, we will study how the depth of nesting of iterated bindings affects the complexity of evaluating queries. An instance of the query evaluation problem consists of a data graph G , an REWB e , a valuation ν for $fv(e)$ and a pair $\langle u, v \rangle$ of nodes in G . The goal is to check if u is connected to v by a data path in $L(e, \nu)$.

4.1 Upper Bounds

An expression in F_i can be thought of as a standard regular expression (without data values) over the alphabet of its sub-expressions. This is the main idea behind our upper bound results. The main result proves that evaluating queries in E_i can be done in Σ_i in the polynomial time hierarchy.

Lemma 4.1. *With an oracle for evaluating E_i queries, F_i queries can be evaluated in polynomial time.*

Proof idea. Suppose the query f_i^1 is to be evaluated on the data graph G and f_i^1 consists of the sub-expressions e_i^1, \dots, e_i^m in E_i . For every j , add an edge labeled e_i^j between those pairs $\langle v_1, v_2 \rangle$ of nodes of G for which $\langle v_1, v_2 \rangle$ is in the evaluation of e_i^j on G . Evaluating the sub-expressions can be done with the oracle. Now f_i^1 can be treated as a standard regular expression over the finite alphabet $\{e_i^1, \dots, e_i^m\}$, and can be evaluated in polynomial time using standard automata theoretic techniques. \square

Theorem 4.2. *For queries in E_i , the evaluation problem belongs to Σ_i .*

Proof idea. Since bindings in E_i are not iterated, each binding is performed at most once. The data value for each variable is guessed non-deterministically. The expression can be treated as a standard regular expression over its sub-expressions and the guessed data values. The sub-expressions are in F_{i-1} , which can be evaluated in polynomial time (Lemma 4.1) with an oracle for evaluating queries in E_{i-1} . This argument will not work in general for arbitrary REWBs — bindings that are nested deeply inside iterations and other bindings may occur more than polynomially many times in a single path. \square

Next we consider the query evaluation problem with the size of the query as the parameter. An instance of the *parameterized weighted circuit satisfiability* problem consists of a Boolean circuit and the parameter $k \in \mathcal{N}$. The goal is to check if the circuit can be satisfied by a truth assignment of weight k (i.e., one that sets exactly k propositional atoms to true). The class $W[P]$ is the set of all parameterized problems which are FPT-reducible to the weighted circuit satisfiability problem.

Theorem 4.3. *Evaluating REWB queries in E_1 , parameterized by the size of the query is in $W[P]$.*

Proof idea. It is proved in [3, Lemma 7, Theorem 8] that a parameterized problem is in $W[P]$ iff there is a non-deterministic Turing machine that takes an instance (x, k) and decides the answer within $f(k)|x|^c$ steps, of which at most $f(k) \log |x|$ are non-deterministic (for some computable function f and a constant c). Such a Turing machine exists for evaluating REWB queries in E_1 , using the steps outlined in the proof idea of Theorem 4.2. \square

Thus, the number of non-deterministic steps needed to evaluate an E_1 query depends only logarithmically on the size of the data graph. It is also known that $W[P]$ is contained in the class $PARA-NP$ — the class of parameterized problems for which there are deterministic algorithms taking instances (x, k) and computing an equivalent instance of the Boolean satisfiability problem in time $f(k)|x|^c$. Hence, we can get an efficient reduction to the satisfiability problem, on which state of the art SAT solvers can be run. Many hard problems in planning fall into this category [7].

We next consider the parameterized complexity of evaluating queries at higher levels. The parameterized class *uniform-XNL* is the class of parameterized problems Q for which there exists a computable function $f : \mathcal{N} \rightarrow \mathcal{N}$ and a non-deterministic algorithm that, given a pair (x, k) , decides if $(x, k) \in Q$ in space at most $f(k) \log |x|$ [3, Proposition 18].

Theorem 4.4. *Evaluating REWB queries, with size of the query as parameter, is in uniform-XNL.*

Proof idea. Let k be the size of the query e_i^1 to be evaluated, on a data graph with n nodes. Suppose a pair of nodes is connected by a data path w in $L(e_i^1)$. Iterations in e_i^1 can only occur inside its F_{i-1} sub-expressions. Hence w consists of at most k sub-paths, each sub-path w_j in the language of some sub-expression f_{i-1}^j . When f_{i-1}^j is considered as a standard regular expression over its sub-expressions (in E_{i-1}), there are no bindings. By a standard pigeon hole principle argument, we can infer that w_j consists of at most kn sub-paths, each one in the language of some sub-expression e_{i-1}^1 . This argument can be continued to prove that w is of length at most $(k^2n)^i$. The existence of such a path can be guessed and verified by a non-deterministic Turing machine in space $O(ik^2 \log n)$. \square

4.2 Lower Bounds

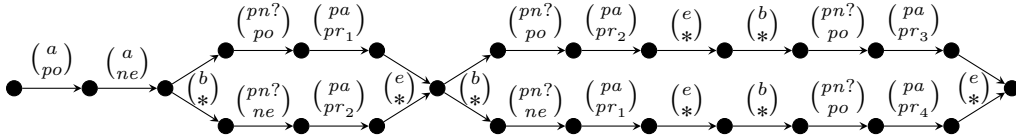
We obtain our lower bounds by reducing various versions of the Boolean formula satisfiability problem to query evaluation. We begin by describing a schema for reducing the problem of evaluating a Boolean formula on a given truth assignment to the problem of evaluating a query on a data graph. The basic ideas for the gadgets we construct below are from [15, proofs of Proposition 2, Theorem 5]. We will need to build on these ideas to address finer questions about the complexity of query evaluation.

Suppose the propositional atoms used in the Boolean formula are among $\{pr_1, \dots, pr_n\}$. We use pr_1, \dots, pr_n also as data values. An edge labeled $\binom{pa}{pr_j}$ indicates the propositional atom pr_j occurring in a sub-formula. The data values po and ne appear on edges labeled with the letter $pn?$, to indicate if a propositional atom appears positively or negatively. The symbol $*$ denotes an arbitrary data value different from all others. We will assume that the Boolean formula is in negation normal form, i.e., negation only appears in front of propositional atoms. This restriction does not result in loss of generality, since any Boolean formula can be converted into an equisatisfiable one in negation normal form with at most linear blowup in the size. The data graph is a series parallel digraph with a source and a sink, defined as follows by induction on the structure of the Boolean formula.

- Positively occurring propositional atom pr_j : $\cdot \xrightarrow{\binom{b}{*}} \cdot \xrightarrow{\binom{pn?}{po}} \cdot \xrightarrow{\binom{pa}{pr_j}} \cdot \xrightarrow{\binom{e}{*}} \cdot$.

- Negatively occurring propositional atom pr_j : $\cdot \xrightarrow{\binom{b}{*}} \cdot \xrightarrow{\binom{pn?}{ne}} \cdot \xrightarrow{\binom{pa}{pr_j}} \cdot \xrightarrow{\binom{e}{*}} \cdot$.
- $\phi_1 \wedge \dots \wedge \phi_r$: inductively construct the data graphs for the conjuncts, then do a standard serial composition, by fusing the sink of one graph with the source of the next one.
- $\phi_1 \vee \dots \vee \phi_r$: inductively construct the data graphs for the disjuncts, then do a standard parallel composition, by fusing all the sources into one node and all the sinks into another node.
- After the whole formula is handled, the source of the resulting graph is fused with the sink of the following graph: $\cdot \xrightarrow{\binom{a}{po}} \cdot \xrightarrow{\binom{a}{ne}} \cdot$.

Let G_ϕ denote the data graph constructed above for formula ϕ . The data graph G_ϕ is shown below for $\phi = (pr_1 \vee \neg pr_2) \wedge ((pr_2 \wedge pr_3) \vee (\neg pr_1 \wedge pr_4))$.



The query uses x_1, \dots, x_k to remember the propositional atoms that are set to true.

$$e_{eval}[k] := a \downarrow_{x_{po}} (a \downarrow_{x_{ne}} ((b(pn?[x_{po}^-] \cdot pa[x_1^- \vee \dots \vee x_k^-] + pn?[x_{ne}^-] \cdot pa[x_1^\neq \wedge \dots \wedge x_k^\neq])e)^*)) . \quad (1)$$

Lemma 4.5. *Let ϕ be a Boolean formula over the propositional atoms pr_1, \dots, pr_n and $\nu : \{x_1, \dots, x_k\} \rightarrow \{pr_1, \dots, pr_n, *\}$ be a valuation. The source of G_ϕ is connected to its sink by a data path in $L(e_{eval}[k], \nu)$ iff ϕ is satisfied by the truth assignment that sets exactly the propositions in $\{pr_1, \dots, pr_n\} \cap \text{Range}(\nu)$ to true.*

Proof idea. The two bindings in the beginning of $e_{eval}[k]$ forces x_{po}, x_{ne} to contain po, ne respectively. A positively occurring propositional atom generates a data path of the form $\cdot \xrightarrow{\binom{b}{*}} \cdot \xrightarrow{\binom{pn?}{po}} \cdot \xrightarrow{\binom{pa}{pr_j}} \cdot \xrightarrow{\binom{e}{*}} \cdot$, which can only be in the language of the expression $b \cdot pn?[x_{po}^-] \cdot pa[x_1^- \vee \dots \vee x_k^-]e$. This forces pr_j to be contained in one of x_1, \dots, x_k . Similar arguments works for negatively occurring atoms. Rest of the proof is by induction on the structure of the formula. \square

Theorem 4.6. *For queries in E_1 , the evaluation problem is NP-hard.*

Proof idea. To check if a Boolean formula ϕ is satisfiable, evaluate the query $a \downarrow_{x_1} a \downarrow_{x_2} \dots a \downarrow_{x_n} e_{eval}[n]$ on the data graph $\cdot \xrightarrow{\binom{a}{pr_1/*}} \cdot \xrightarrow{\binom{a}{pr_2/*}} \dots \xrightarrow{\binom{a}{pr_n/*}} \cdot - G_\phi \rightarrow \cdot$. Here, $\xrightarrow{\binom{a}{pr_j/*}}$ denotes two edges in parallel, one labeled with $\binom{a}{pr_j}$ and another with $\binom{a}{*}$. \square

Evaluating queries in E_1 is NP-complete, evaluating REWB queries in general is PSPACE-complete and evaluating queries in E_i is in Σ_i . To prove a corresponding Σ_i lower bound, one would need to simulate Σ_i computations using queries with bounded depth of nesting of iterated bindings. However, this does not seem to be possible. We take a closer look at this in the rest of the paper. Finding the exact complexity of evaluating queries in E_i remains open.

We now extend our satisfiability-to-query evaluation schema to handle Boolean quantifiers. Let $PR = \{pr_1, \dots, pr_n\}$ be a set of propositional atoms. To handle existential Boolean quantifiers, we build a new graph and a query. These gadgets build on earlier ideas to bring out the difference in the role played by the data graph and the query while reducing satisfiability to query evaluation. The new graph $G[\exists k/PR] \circ G$, is as follows: $\cdot \xrightarrow{\binom{a_1}{pr_1}} \cdot \xrightarrow{\binom{a_1}{pr_2}} \dots \xrightarrow{\binom{a_1}{pr_n}}$

The graph G_0 and the expression e^0 are designed to ensure that the source of G is connected to its sink by a path in $L(e, \nu)$, unless ν is not injective, in which case G can be bypassed by one of the edges labeled $\binom{skip}{pr_j}$ introduced in G_0 . The graph G_i and the expression e_i are designed to ensure that any path from the source of G_i to its sink has to go through G_{i-1} multiple times, once for each pr_j stored in the variable x_i . The nesting depth of iterated bindings in the expression e^i is one more than that of e^{i-1} .

Suppose ν is a partial valuation of some variables, whose domain does not intersect with $\{x_1, \dots, x_k\}$. We denote by $\nu[\{x_1, \dots, x_k\} \rightarrow PR]$ the set of valuations ν' that extend ν such that $\text{domain}(\nu') = \text{domain}(\nu) \cup \{x_1, \dots, x_k\}$ and $\{\nu'(x_1), \dots, \nu'(x_k)\} \subseteq PR$. We additionally require that ν' is injective on $\{x_1, \dots, x_k\}$ when we write $\nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$.

Lemma 4.9. *Let $i \in \{1, \dots, k\}$ and ν_i be a valuation for $\text{fv}(e^i) \setminus \{x_1, \dots, x_i\}$. The source of G_i is connected to its sink by a data path in $L(e^i, \nu_i)$ iff for every $\nu \in \nu_i[\{x_1, \dots, x_i\} \rightarrow PR]$, there is a data path in $L(e^0, \nu)$ connecting the source of G_0 to its sink.*

Proof idea. The data path has to begin with $b_i \binom{a_i}{pr_1}$ in the language of $b_i a_i \downarrow_{x_i}$, forcing x_i to store pr_1 . Then the path has to traverse G_{i-1} using e^{i-1} . At the sink of G_{i-1} , the path is forced to take $\binom{a_i}{pr_1} c_i$ to satisfy the condition in $a_i[x_i^-] c_i$. This forces the path to start again in $\binom{a_i}{pr_2}$ and so on. \square

We write $G[\forall k/PR] \circ G$ and $e[\forall k] \circ e$ to denote the graph G_k and REWB e^k constructed above. We implicitly assume that the variables x_1, \dots, x_k are not bound inside e . We can always rename variables to ensure this. If e is in E_i , then $e[\forall k] \circ e$ is in F_{i+k-1} .

Lemma 4.10. *Let ν be a valuation for $\text{fv}(e) \setminus \{x_1, \dots, x_k\}$ for some REWB e . The source of $G[\forall k/PR] \circ G$ is connected to its sink by a data path in $L(e[\forall k] \circ e, \nu)$ iff for all $\nu' \in \nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$, the source of G is connected to its sink by a data path in $L(e, \nu')$.*

Proof idea. Lemma 4.9 ensures that there is a path $w_{\nu'}$ in $L(e^0, \nu')$ connecting the source of G_0 to its sink for every valuation $\nu' \in \nu[\{x_1, \dots, x_k\} \rightarrow PR]$. From Figure 1, $w_{\nu'}$ can either be a *skip* edge, or a path through G . By definition, e^0 allows a skip edge to be taken only when two variables among x_1, \dots, x_k have the same data value. Hence for valuations ν' that are injective on $\{x_1, \dots, x_k\}$, $w_{\nu'}$ is in $L(e, \nu')$. \square

If ϕ is a partially quantified Boolean formula with the propositional atoms in PR occurring freely, we write $\exists^k PR \phi$ to denote that atoms in PR are existentially quantified with the constraint that exactly k of them should be set to true. We write $\forall^k PR \phi$ to denote that atoms in PR are universally quantified and that only those assignments that set exactly k of the atoms to true are to be considered. An instance of the *weighted quantified satisfiability* problem consists of a Boolean formula ϕ over the set PR of propositional atoms, a partition PR_1, \dots, PR_ℓ of PR and numbers k_1, \dots, k_ℓ . The goal is to check if $(\exists^{k_1} PR_1 \forall^{k_2} PR_2 \dots \phi)$ is true.

Lemma 4.11. *Given an instance of the weighted quantified satisfiability problem, We can construct in polynomial time a data graph G and an REWB $e_{1+k_2+k_4+\dots}^1$ satisfying the following conditions.*

1. *The source of G is connected to its sink by a data path in $L(e_{1+k_2+k_4+\dots}^1)$ iff the given instance of the weighted quantified satisfiability problem is a yes instance.*
2. *The size of $e_{1+k_2+k_4+\dots}^1$ depends only on k_1, \dots, k_ℓ .*

Proof idea. The required data graph G is $G[\exists k_1/PR_1] \circ G[\forall k_2/PR_2] \circ \dots \circ G_\phi$ and the required REWB $e_{1+k_2+k_4+\dots}^1$ is $e[\exists k_1] \circ e[\forall k_2] \circ \dots \circ e_{eval}[k_1 + \dots + k_\ell]$. We assume that \circ associates to the right, so $G_1 \circ G_2 \circ G_3$ is $G_1 \circ (G_2 \circ G_3)$ and $e^1 \circ e^2 \circ e^3$ is $e^1 \circ (e^2 \circ e^3)$. Correctness follows from Lemma 4.10 and Lemma 4.5. \square

The weighted quantified satisfiability problem is parameterized by $\ell + k_1 + \dots + k_\ell$. The class $\text{AW}[\text{SAT}]$ is the set of parameterized problems that are FPT-reducible to the weighted quantified satisfiability problem (see [8, Chapter 26]).

Theorem 4.12. *Evaluating REWB queries, parameterized by the size of the query is hard for AW[SAT] under FPT reductions.*

Proof. The reduction given in Lemma 4.11 is a FPT reduction from the weighted quantified satisfiability problem to the problem of evaluating REWB queries, with query size as the parameter. \square

5 Summary and Open Problems

We have proved that increasing the depth of nesting of iterated bindings in REWBs increase expressiveness. Given an REWB, it is undecidable to check if its language can be defined with another REWB with smaller depth of nesting of iterated bindings. The complexity of query evaluation problems are summarized in the following table, followed by a list of technical challenges to be overcome for closing the gaps.

Query level	Evaluation	Parameterized complexity, query size is parameter
E_1	NP-complete	(?2)W[SAT] lower bound, W[P] upper bound
$E_i, i > 1$	(?1), Σ_i upper bound	(?3)
Unbounded	PSPACE-complete [15]	(?4)AW[SAT] lower bound, uniform-XNL upper bound

1. Suppose we want to check the satisfiability of a Σ_2 Boolean formula over $(n_e + n_u)$ propositional atoms of which the first n_e atoms are existentially quantified and the last n_u are universally quantified. With currently known techniques, reducing this to query evaluation results in an REWB in $E_{(n_u+1)}$. Hence, with bounded nesting depth, we cannot even prove a Σ_2 lower bound.
2. Weighted formula satisfiability, complete for W[SAT], can be simulated with series-parallel graphs. Queries in E_1 do not seem to be powerful enough for weighted circuits.
3. Without parameterization, the Σ_i upper bound is obtained by an oracle hierarchy of NP machines. With parameterization, an oracle hierarchy of W[P] machines does not correspond to any parameterized complexity class. See [3, Section 4] for discussions on subtle points which make classical complexity results fail in parameterized complexity.
4. As in point 2, here one might hope for a AW[P] lower bound, which is quantified weighted circuit satisfiability (stronger than AW[SAT], which is quantified weighted formula satisfiability). Even if this improvement can be made, there is another classical complexity result not having analogous result in parameterized complexity: not much is known about the relation between parameterized alternating time bounded class (AW[P]) and parameterized space bounded class (uniform-XNL).

Acknowledgements The authors thank Partha Mukhopadhyay and Geevarghese Philip for helpful discussions about polynomial time hierarchy and parameterized complexity theory.

References

- [1] P. Barceló. Querying graph databases. In *Proceedings of PODS*, pages 175–188, New York, NY, USA, 2013. ACM.
- [2] P. Barceló, J. Reutter, and L. Libkin. Parameterized regular expressions and their languages. *Theoretical Computer Science*, 474:21–45, 2013.
- [3] Yijia Chen, J. Flum, and M. Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *Computational Complexity, 2003*, pages 13–29, 2003.

- [4] T. Colcombet and A. Manuel. Generalized data automata and fixpoint logic. In *FSTTCS*, volume 29 of *LIPIcs*, pages 267–278, 2014.
- [5] T. Colcombet and A. Manuel. Combinatorial expressions and lower bounds. In *STACS*, volume 30 of *LIPIcs*, pages 249–261, 2015.
- [6] T. Colcombet and A. Manuel. Fragments of fixpoint logic on data words. In *FSTTCS*, volume 45 of *LIPIcs*, pages 98–111, 2015.
- [7] R. De Haan, M. Kronegger, and A. Pfandler. Fixed-parameter tractable reductions to sat for planning. In *Proceedings of IJCAI*, pages 2897–2903, 2015.
- [8] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Thomson Brooks/Cole, 1997.
- [9] L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10(4):385–397, 1963.
- [10] O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *LATA*, volume 6031 of *LNCS*, pages 561–572, 2010.
- [11] C. Gutierrez, C. Hurtado, and A. Mendelzon. Foundations of semantic web databases. *JCSS*, 77(3):520–541, 2011.
- [12] E.V. Kostylev, J.L. Reutter, and D. Vrgoč. Containment of data graph queries. In *ICDT*, pages 131–142, 2014.
- [13] U. Leser. A query language for biological networks. *Bioinformatics*, 21(suppl 2):ii33–ii39, 2005.
- [14] L. Libkin, W. Martens, and D. Vrgoč. Querying graph databases with xpath. In *ICDT*, pages 129–140, New York, NY, USA, 2013. ACM.
- [15] L. Libkin, T. Tan, and D. Vrgoč. Regular expressions with binding over data words for querying graph databases. In *DLT*, volume 7907 of *LNCS*, pages 325–337, 2013.
- [16] L. Libkin and D. Vrgoč. Regular path queries on graphs with data. In *ICDT*, pages 74–85, 2012.
- [17] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.
- [18] W3C Recommendation. Sparql 1.1 query language. 21 March 2013.
- [19] R. Ronen and O. Shmueli. Soql: A language for querying and creating data in social networks. In *ICDE*, pages 1595–1602, 2009.
- [20] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, volume 4207 of *LNCS*, pages 41–57, 2006.
- [21] M. Sipser. *Introduction to the Theory of Computation*. Springer, 2013.
- [22] T. Tan. Graph reachability and pebble automata over infinite alphabets. *ACM Trans. Comput. Logic*, 14(3):19:1–19:31, 2013.
- [23] D. Vrgoč. Using variable automata for querying data graphs. *Information Processing Letters*, 115(3):425–430, 2015.

A Details of Section 3

This section contains complete proofs and explanations from Section 3. We start with the semantics of the automaton view of expressions.

A.1 Automata View of Expressions

We will now provide in more detail the semantics of the automata defined for each expression. Expressions we consider contain some free variables and some bound variables due to the \downarrow_x operator. Without loss of generality, we will assume that no two occurrences of the binding operator contain the same variable name. Recall that for an expression e , we denote the set of its free variables by $fv(e)$, and the set of all variables (free and bound) by $var(e)$. A valuation associates every free variable to a data value.

Consider an expression f_i^1 , and its corresponding automaton $\mathcal{A}(f_i^1)$. Let ν be a valuation which associates a data value to all the free variables $fv(f_i^1)$ of f_i^1 . A run of $\mathcal{A}(f_i^1)$ over a data word $w = \binom{a_1}{d_1} \binom{a_2}{d_2} \dots \binom{a_n}{d_n}$ given valuation ν is as follows:

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_{m-1}} q_{m-1} \xrightarrow{w_m} q_m$$

where:

- q_0 is an initial state,
- $w = w_1 w_2 \dots w_m$,
- if $i = 0$ then $m = n$ and for each j , we have $w_j = \binom{a_j}{d_j}$. Moreover, for each j , there exists a transition $q_{j-1} \xrightarrow{a_j} q_j$ or $q_{j-1} \xrightarrow{a_j[c]} q_j$ such that $d_j, \nu \models c$,
- if $i > 0$, then for each j there exists a transition $q_{j-1} \xrightarrow{e_i^j} q_j$ such that $w_j \in L(e_i^j, \nu \upharpoonright e_i^j)$, where $\nu \upharpoonright e_i^j$ denotes the valuation restricted to $fv(e_i^j)$.

The run is accepting if q_m is an accepting state of the automaton. The language $L(\mathcal{A}(f_i^1), \nu)$ is the set of words for which $\mathcal{A}(f_i^1)$ has an accepting run given valuation ν .

Given an expression e_i^1 and a valuation ν of its free variables, the run of $\mathcal{A}(e_i^1)$ on a data word w is defined as:

$$(q_0, \nu_0) \xrightarrow{w_1} (q_1, \nu_1) \xrightarrow{w_2} \dots \xrightarrow{w_{m-1}} (q_{m-1}, \nu_{m-1}) \xrightarrow{w_m} (q_m, \nu_m)$$

where

- q_0 is an initial state,
- each w_j is a data word such that $w = w_1 w_2 \dots w_m$,
- each ν_j is a partial function from $var(e_i^1)$ to the set of data values, with $\nu_0 = \nu$;
- for each j , either $w_j = \binom{a}{d}$ and there is a transition $q_{j-1} \xrightarrow{a \downarrow_x} q_j$ and $\nu_j = \nu_{j-1}[x \rightarrow d]$, or there is a transition $q_{j-1} \xrightarrow{f_{i-1}^j} q_j$ with $w_j \in L(f_{i-1}^j, \nu_{j-1} \upharpoonright f_{i-1}^j)$ and $\nu_j = \nu_{j-1}$. As before, $\nu_{j-1} \upharpoonright f_{i-1}^j$ is a valuation for f_{i-1}^j obtained by restricting the partial function ν_{j-1} to $fv(f_{i-1}^j)$.

The notion of acceptance and language $L(\mathcal{A}(e_i^1), \nu)$ are defined in a way similar to the F_i case. We now explain with an example the necessity of the restriction that no two occurrences of the binding operator contain the same variable name. Suppose $e_1^1 = a \downarrow_x (b \downarrow_x (c[x=]) \cdot c[x\neq])$. An automaton would have the transitions $q_0 \xrightarrow{a \downarrow_x} q_1 \xrightarrow{b \downarrow_x} q_2 \xrightarrow{c[x=]} q_3 \xrightarrow{c[x\neq]} q_4$. There is no elegant way to specify that the value to be tested in the

transition $q_3 \xrightarrow{c[x^\neq]} q_4$ is the one bound in $q_0 \xrightarrow{a \downarrow_x} q_1$ and not the one bound in $q_1 \xrightarrow{b \downarrow_x} q_2$. Hence we consider the language equivalent expression $a \downarrow_{x_1} (b \downarrow_{x_2} (c[x_2^\neq]) \cdot c[x_1^\neq])$, which avoids this problem.

The following lemma can be shown by an induction on i .

Lemma A.1. *For every expression f_i^1 , and for every valuation ν of $fv(f_i^1)$, the languages $L(f_i^1, \nu)$ and $L(\mathcal{A}(f_i^1), \nu)$ are equal. Similarly for expressions e_i^1 .*

For any expression e , the size of $\mathcal{A}(e)$ is defined as the number of states present.

A.2 Strictness of the Hierarchy

Using the semantics of the automata developed above, we will give a full proof of Lemma 3.2.

Lemma 3.2. Let e_i^1 be an expression and let $n \in \mathcal{N}$ be greater than $(|\mathcal{A}(e)| + 1)$ and $(|var(e)| + 1)$ for every sub-expression e of e_i^1 . Let ν be a valuation of $fv(e_i^1)$ and let x, z be data words. Then: $xu_{i,n}z \in L(e_i^1, \nu) \implies x\bar{u}_{i,n}z \in L(e_i^1, \nu)$ for some $\bar{u}_{i,n} \in Mismatch_{i,n}$.

Proof. We proceed by an induction on i . We start with the base case. Suppose $xu_{1,n}z \in L(e_1^1, \nu)$ for some expression e_1^1 with $\max(|\mathcal{A}(e_1^1)|, |var(e_1^1)|) < n$. The automaton $\mathcal{A}(e_1^1)$ has an accepting run of the following form:

$$\text{Run } \rho_1 : (q_0, \nu_0) \xrightarrow{w_1} (q_1, \nu_1) \xrightarrow{w_2} \dots \xrightarrow{w_{m-1}} (q_{m-1}, \nu_{m-1}) \xrightarrow{w_m} (q_m, \nu_m)$$

where $xu_{1,n}z = w_1w_2 \dots w_m$. Recall that automata for E_1 -expressions are acyclic, so states cannot repeat in a run. Since the number of states is strictly less than n and $u_{1,n}$ contains n^2 occurrences of a_1b_1 , there is some w_p which contains n occurrences of a_1b_1 :

$$\binom{a_1}{d[1, j+1]} \binom{b_1}{d[1, j+1]} \dots \binom{a_1}{d[1, j+n]} \binom{b_1}{d[1, j+n]}$$

Then by definition of runs of $\mathcal{A}(e_1^1)$:

- there is a transition $q_{p-1} \xrightarrow{f_0^1} q_p$ in $\mathcal{A}(e_1^1)$ with $w_p \in L(f_0^1, \nu_{p-1})$ and
- valuation ν_p equals ν_{p-1} since the transition $q_{p-1} \rightarrow q_p$ does not contain a binding.

Note that $Range(\nu_{p-1})$ can contain at most $|var(e_1^1)|$ distinct data values. Since by assumption n is strictly bigger $|var(e_1^1)|$, there can be at most $n - 1$ distinct data values in $Range(\nu_{p-1})$.

Let us now zoom in to the accepting run of $\mathcal{A}(f_0^1)$ on the sub-word w_p .

$$\text{Run } \sigma_1 : q'_0 \xrightarrow{w'_1} q'_1 \xrightarrow{w'_2} \dots \xrightarrow{w'_{s-1}} q'_{s-1} \xrightarrow{w'_s} q'_s$$

with $w_p = w'_1w'_2 \dots w'_s$. Each transition reads a single letter: that is, it is of the form $q'_{j-1} \xrightarrow{a} q'_j$ or $q_{j-1} \xrightarrow{a[c]} q_j$ with letter a denoting either a_1 or b_1 . Note that there can be no further bindings in this level F_0 . So, each condition $a[c]$ in a transition can check for equality or inequality with respect to data values in $Range(\nu_{p-1})$. Consider the transitions reading b_1 . As there are at most $n - 1$ data values in $Range(\nu_{p-1})$, there is some $\binom{b_1}{d[1, j']}$ in w_p such that $d[1, j']$ is different from all values in $Range(\nu_{p-1})$. Therefore, changing $d[1, j']$ in $\binom{b_1}{d[1, j']}$ to a new data value $d'[1, j'] \notin Range(\nu_{p-1})$ will give a data word which continues to satisfy all conditions occurring in the run σ_1 of $\mathcal{A}(f_0^1)$. The run ρ_1 is oblivious to this change. This is because there are no bindings in σ_1 and hence the valuation ν_p is the same as ν_{p-1} . Hence the same run ρ_1 of $\mathcal{A}(e_1^1)$ accepts this modified word. Observe that in this word there is a mismatch between an a_1 and the consecutive b_1 occurring in $u_{1,n}$ and is of the form $x\bar{u}_{1,n}z$ as required by the lemma. This proves the lemma for the base case $i = 1$.

We will now prove the induction step. Assume that the lemma is true for some $i - 1$. We will now prove it for i . Consider the word $xu_{i,n}z$. Suppose it belongs to $L(e_i^1, \nu)$ for some expression e_i^1 with the value n being an upper bound on $|\mathcal{A}(e)| + 1$ and $|\text{var}(e)| + 1$ for every subexpression e of e_i^1 . Let ρ_i be the accepting run of $\mathcal{A}(e_i^1)$ on $xu_{i,n}z$:

$$\text{Run } \rho_i : (q_0, \nu_0) \xrightarrow{w_1} (q_1, \nu_1) \xrightarrow{w_2} \dots \xrightarrow{w_{m-1}} (q_{m-1}, \nu_{m-1}) \xrightarrow{w_m} (q_m, \nu_m)$$

with $xu_{i,n}z = w_1w_2 \dots w_m$. Since the automaton $\mathcal{A}(e_i^1)$ is acyclic, no state can repeat in ρ_i . As the number of states is less than n and $u_{i,n}$ contains n^2 occurrences of $a_iu_{i-1,n}b_i$, some w_p contains n occurrences of the block $a_iu_{i-1,n}b_i$:

$$\left(\begin{smallmatrix} a_i \\ d[i, j + 1] \end{smallmatrix} \right)^{u_{i-1,n}} \left(\begin{smallmatrix} b_i \\ d[i, j + 1] \end{smallmatrix} \right) \dots \left(\begin{smallmatrix} a_i \\ d[i, j + n] \end{smallmatrix} \right)^{u_{i-1,n}} \left(\begin{smallmatrix} b_i \\ d[i, j + n] \end{smallmatrix} \right)$$

Then, by definition of runs of $\mathcal{A}(e_i^1)$:

- there is a transition $q_{p-1} \xrightarrow{f_{i-1}^1} q_p$ in $\mathcal{A}(e_i^1)$ with $w_p \in L(f_{i-1}^1, \nu_{p-1})$ and
- $\nu_p = \nu_{p-1}$.

As $\text{Range}(\nu_{p-1})$ can contain at most $|\text{var}(e_i^1)|$ distinct data values, and since n is bigger than $|\text{var}(e_i^1)| + 1$, we observe that $\text{Range}(\nu_{p-1})$ contains at most $n - 1$ distinct data values. Consider the run of $\mathcal{A}(f_{i-1}^1)$ on w_p given valuation ν_{p-1} :

$$\text{Run } \sigma_i : q'_0 \xrightarrow{w'_1} q'_1 \xrightarrow{w'_2} \dots \xrightarrow{w'_{s-1}} q'_{s-1} \xrightarrow{w'_s} q'_s$$

where $w_p = w'_1w'_2 \dots w'_s$.

If some w'_j contains $u_{i-1,n}$ entirely, then this w'_j belongs to the language of $L(e_{i-1}^1, \nu_{p-1})$ for some expression e_{i-1}^1 . This is as per the definition of runs of F_i automata. Additionally, e_{i-1}^1 is a subexpression of e_i^1 and hence satisfies the condition that n is bigger than $|\mathcal{A}(e)| + 1$ and $|\text{var}(e)| + 1$ for every subexpression e of e_{i-1}^1 . We can then use the induction hypothesis to infer that there is a mismatched word in $L(e_{i-1}^1, \nu_{p-1})$. Hence we can replace w'_j with this mismatched word to obtain the same runs σ_i and ρ_i , thus proving the lemma for this case.

Otherwise, no w'_j contains both $\left(\begin{smallmatrix} a_i \\ d[i, j] \end{smallmatrix} \right)$ and $\left(\begin{smallmatrix} b_i \\ d[i, j] \end{smallmatrix} \right)$ of a block $a_iu_{i-1,n}b_i$. As $\text{Range}(\nu_{p-1})$ has at most $n - 1$ distinct data values, there is one $\left(\begin{smallmatrix} b_i \\ d[i, j] \end{smallmatrix} \right)$ such that $d[i, j]$ is not present in $\text{Range}(\nu_{p-1})$. Let this be present in w'_k , and let e_{i-1}^k be the sub-expression in the transition $q'_{k-1} \rightarrow q'_k$ with $w'_k \in L(e_{i-1}^k, \nu_{p-1})$. Consider a fresh data value $d' \notin \text{Range}(\nu_{p-1})$ and which is different from every data value in w'_k . Change all occurrences of $d[i, j]$ in w'_k to this fresh data value d' . The modified word belongs to $L(e_{i-1}^k, \nu_{p-1})$ (this can be shown by a structural induction for general REWBs). Therefore the run σ_i holds for the word w_p with w'_k modified to this new word. Moreover, as discussed in the base case, the run ρ_i is not affected by this change as the valuation ν_p is the same as ν_{p-1} . This gives a word with a mismatch between an a_i and the corresponding b_i that is accepted by e_i^1 , thereby proving the lemma. □

A.3 Undecidability of Membership at a Given Level

This section is devoted to proof of the following theorem.

Theorem 3.4. Given an expression in F_{i+1} , checking if there exists an equivalent expression in F_i is undecidable.

The basic idea is from the proof of undecidability of universality of REWBs and related formalisms [17, 15]. If a given REWB is universal, i.e., accepts all data words, then there is a language equivalent expression that does not use any binding. The undecidability of universality can hence be interpreted to mean that determining

the usefulness of bindings in an expression is undecidable. We combine this insight with results we have obtained for the expressions r_i in the previous sub-section to prove Theorem 3.4. We proceed by a reduction from Post's Correspondence Problem (PCP). An instance of PCP is a set $\{(u_1, v_1), (u_2, v_2), \dots, (u_n, v_n)\}$ of pairs of words over a finite alphabet Σ_{PCP} . A solution to this instance is a sequence l_1, l_2, \dots, l_m with each $l_j \in \{1, \dots, n\}$ such that $u_{l_1} u_{l_2} \dots u_{l_m} = v_{l_1} v_{l_2} \dots v_{l_m}$.

Suppose we are given an instance $\{(u_1, v_1), (u_2, v_2), \dots, (u_n, v_n)\}$ of PCP. We will encode a solution l_1, \dots, l_m to this instance by a set of data words of the form:

$$\theta_1 \begin{pmatrix} \# \\ d_1 \end{pmatrix} z \begin{pmatrix} \# \\ d_2 \end{pmatrix} \theta_2$$

where:

- $z \in L(r_i)$, with r_i being the expression in Definition 3.1,
- θ_1 is the data word:

$$\begin{pmatrix} \$_{l_1} \\ h_1 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ 1 \end{pmatrix} \dots \begin{pmatrix} \alpha_p \\ p \end{pmatrix} \begin{pmatrix} \$_{l_2} \\ h_2 \end{pmatrix} \begin{pmatrix} \alpha_{p+1} \\ p+1 \end{pmatrix} \dots \begin{pmatrix} \$_{l_m} \\ h_m \end{pmatrix} \dots \begin{pmatrix} \alpha_r \\ r \end{pmatrix}$$

where the word $\alpha_s \dots \alpha_t$ between $\$_{l_j}$ and $\$_{l_{j+1}}$ equals the word u_{l_j} ,

- θ_2 is the data word:

$$\begin{pmatrix} \$_{l_1} \\ h_1 \end{pmatrix} \begin{pmatrix} \beta_1 \\ 1 \end{pmatrix} \dots \begin{pmatrix} \beta_q \\ q \end{pmatrix} \begin{pmatrix} \$_{l_2} \\ h_2 \end{pmatrix} \begin{pmatrix} \beta_{q+1} \\ q+1 \end{pmatrix} \dots \begin{pmatrix} \$_{l_m} \\ h_m \end{pmatrix} \dots \begin{pmatrix} \beta_r \\ r \end{pmatrix}$$

where the word $\beta_s \dots \beta_t$ between $\$_{l_j}$ and $\$_{l_{j+1}}$ equals the word v_{l_j} ,

- the data values $\{1, \dots, r, d_1, d_2, h_1, \dots, h_m\}$ are all distinct.

We will first construct an expression Δ that accepts all words of the form $w_1 \begin{pmatrix} \# \\ d_1 \end{pmatrix} z \begin{pmatrix} \# \\ d_2 \end{pmatrix} w_2$ with $z \in L(r_i)$ such that the part $w_1 \# \# w_2$ does not satisfy the conditions mentioned above. The expression Δ will be in E_{i+1} . We will then reason that this expression will have an equivalent expression in F_i iff PCP has no solution.

Let Γ denote the finite alphabet $\Sigma_{PCP} \cup \{\$, \dots, \$n, \#\}$. We will now exhaustively reason about the situations when a word $w_1 \begin{pmatrix} \# \\ d_1 \end{pmatrix} z \begin{pmatrix} \# \\ d_2 \end{pmatrix} w_2$ is not an encoding of the PCP solution. This will give us the expression Δ mentioned above.

- Projection of the word on to the finite alphabet is not of the form $(\$_1 u_1 + \dots + \$n u_n)^* \# z \# (\$1 v_1 + \dots + \$n v_n)^*$. Let ϕ_1 and ϕ_2 be the regular expressions denoting the complement of $(\$_1 u_1 + \dots + \$n u_n)^*$ and $(\$1 v_1 + \dots + \$n v_n)^*$ respectively. The required expression that accepts words with a mistake in the finite alphabet is $\phi_1 \# r_i \# \Gamma^* + \Gamma^* \# r_i \# \phi_2$. Note that this expression is at the same level as r_i since r_i is not in the scope of any binding.

- Words where the data values are not according to the encoding. Firstly, words of the form $\dots \begin{pmatrix} \# \\ d \end{pmatrix} z \begin{pmatrix} \# \\ d' \end{pmatrix} \dots$ where d or d' repeat. Expression accepting words where d repeats is given by:

$$\bigvee_{a \in \Gamma} (\Gamma^* a \downarrow_x (\Gamma^* \# [x^=]) r_i \Gamma^* + \Gamma^* \# \downarrow_x (r_i \Gamma^* a [x^=] \Gamma^*))$$

A similar expression can be given for the case where d' repeats. Since these expressions add a binding over r_i , they are in E_{i+1} .

- Words of the form $\dots \begin{pmatrix} * \\ d \end{pmatrix} \dots \begin{pmatrix} * \\ d \end{pmatrix} \dots \# z \# \dots$ where a data value repeats before the $\# z \#$ and words of the form $\dots \# z \# \dots \begin{pmatrix} * \\ d \end{pmatrix} \dots \begin{pmatrix} * \\ d \end{pmatrix} \dots$ where values repeat after $\# z \#$

$$\bigvee_{a, b \in \Gamma} (\Gamma^* a \downarrow_x (\Gamma^* b [x^=] \Gamma^*) \# r_i \# \Gamma^* + \Gamma^* \# r_i \# \Gamma^* a \downarrow_x (\Gamma^* b [x^=] \Gamma^*))$$

• Note that in the encoding of the solution, the data values in the j^{th} dollar symbol before and after $\# z \#$ need to be the same. We will now consider words where this is not true. Let us first look at words where the mismatch occurs either in the first dollar symbol or in the last dollar symbol.

$$\bigoplus_{\$, \$' \in \{\$, \dots, \$n\}} (\$ \downarrow_x (\Gamma^* \# r_i \# \$'[x^\neq]) \Gamma^* + \Gamma^* \$ \downarrow_x (\Sigma_{PCP}^* \# r_i \# \Gamma^* \$'[x^\neq]) \Sigma_{PCP}^*)$$

Suppose the first dollar symbols to the left and right of $\# z \#$ have the same data value, and so do the last dollar symbols. In this case, if there some j such that the j^{th} dollar symbol to the left and right of $\# z \#$ have different data values, the data word is of the following form.

$$\dots \binom{\delta_1}{d} \Sigma_{PCP}^* \binom{\delta_2}{d'} \dots \# z \# \dots \binom{\delta_3}{d} \Sigma_{PCP}^* \binom{\delta_4}{d''} \dots$$

where $\delta_1, \delta_2, \delta_3, \delta_4 \in \{\$, \dots, \$n\}$ and $d' \neq d''$. There is a data value d occurring with a dollar on both sides, and the data values attached with next dollar symbols on the two sides do not match. The expression for such words is given by:

$$\bigoplus_{\delta_1, \dots, \delta_4 \in \{\$, \dots, \$n\}} (\Gamma^* \delta_1 \downarrow_x (\Sigma_{PCP}^* \delta_2 \downarrow_y (\Gamma^* \# r_i \# \Gamma^* \delta_3[x^\neq] \Sigma_{PCP}^* \delta_4[y^\neq])) \Gamma^*)$$

• Now we will consider words where the mismatch of data values occurs in a non-dollar position. We start with the expression for words with a mismatch in the first or last non-dollar position:

$$\bigoplus_{\delta_1, \delta_2 \in \{\$, \dots, \$n\}, a, b \in \Sigma_{PCP}} \delta_1 a \downarrow_x (\Gamma^* \# r_i \# \delta_2 b[x^\neq]) \Gamma^* + \Gamma^* a \downarrow_x (\# r_i \# \Gamma^* b[x^\neq])$$

For detecting mismatch at an intermediate position, we resort to the same idea as in the previous case. We consider words of the form:

$$\dots \binom{\alpha_1}{d} (\varepsilon + \delta_1) \binom{\alpha_2}{d'} \dots \# r_i \# \dots \binom{\alpha_3}{d} (\varepsilon + \delta_2) \binom{\alpha_4}{d''} \dots$$

The expression for such words is given by:

$$\bigoplus_{\delta_1, \delta_2 \in \{\$, \dots, \$n\}, \alpha_1, \dots, \alpha_4 \in \Sigma_{PCP}} \Gamma^* \alpha_1 \downarrow_x ((\varepsilon + \delta_1) \alpha_2 \downarrow_y (\Gamma^* \# r_i \# \Gamma^* \alpha_3[x^\neq] (\varepsilon + \delta_2) \alpha_4[y^\neq])) \Gamma^*$$

• We are now left with words where the data values on every corresponding position before and after $\# z \#$ match. Among these words, the non-solutions are the ones where for a particular data value occurring on both sides of $\# z \#$, the corresponding letters do not match. The expression for such words is given by:

$$\bigoplus_{\gamma_1 \neq \gamma_2} \Gamma^* \gamma_1 \downarrow_x (\Gamma^* \# r_i \# \Gamma^* \gamma_2[x^\neq]) \Gamma^*$$

The required expression Δ is the sum of all the above expressions. Note that Δ has a binding made on the left side of $\# r_i \#$ that is checked on the right side. This makes expression Δ to fall in E_{i+1} as expression r_i is in F_i .

Lemma A.2. *The expression Δ has an equivalent expression in F_i iff the given PCP instance has no solution.*

Proof. Suppose PCP instance has no solution. Then all words of the form $w_1 \# z \# w_2$ with $w_1, w_2 \in \Gamma^*$ and $z \in L(r_i)$ are in the language of the expression Δ . Therefore an equivalent expression for Δ is $\Gamma^* \# r_i \# \Gamma^*$. This expression is in F_i as the expression r_i is in F_i .

Suppose PCP instance has a solution. Let us assume that Δ has an equivalent expression f_i^1 . We will show that this leads to a contradiction. For technical convenience, let us assume that f_i^1 has no free variables (the case with

free variables can be handled in a similar way). Let n be a natural number such that $|\mathcal{A}(f_i^1)| < n$. Consider a word $\theta_1 \# u_{i,n} \# \theta_2$ that encodes the solution of the PCP instance. Let θ'_2 be a new data word obtained from θ_2 by modifying the last data value to a fresh data value not occurring in $\theta_1 \# u_{i,n} \# \theta_2$. Then, $\theta_1 \# u_{i,n} \# \theta'_2$ does not encode any solution and hence belongs to $L(\Delta)$. Let us now look at the run of $\mathcal{A}(f_i^1)$ on the word $\theta_1 \# u_{i,n} \# \theta'_2$:

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_{m-1}} q_{m-1} \xrightarrow{w_m} q_m$$

where $w_1 w_2 \dots w_m = \theta_1 \# u_{i,n} \# \theta'_2$.

If some w_p contains $u_{i,n}$ entirely, then by definition of runs of $\mathcal{A}(f_i^1)$, there is some subexpression e_i^1 and a word $xu_{i,n}z$ such that $xu_{i,n}z \in L(e_i^1)$. Note that we assumed that there are no free variables. Then, by Lemma 3.2 there is a word $x\bar{u}_{i,n}z \in L(e_i^1)$ where $\bar{u}_{i,n}$ contains a mismatch. However, by definition of Δ , this is not possible. Therefore no w_p can contain $u_{i,n}$ entirely. This would then imply that θ_1 and θ'_2 lie in different w_j : in particular, the last part of the run w_m contains the last letter in θ'_2 and moreover does not contain any part of θ_1 . Hence, data values in θ'_2 are never compared with those in θ_1 . Note that by the definition of runs, the word $w_m \in L(e_i^2)$ for some subexpression e_i^2 . Changing the last data value of w_m back to the value in θ_2 will result in a word which is an automorphic copy of w_m and hence this modified word should also lie in $L(e_i^2)$. This shows that the same run of $\mathcal{A}(f_i^1)$ can accept $\theta_1 \# u_{i,n} \# \theta_2$ which encodes a solution of the PCP instance. Therefore the expression supposed to be equivalent to Δ accepts a solution of the PCP instance. A contradiction. \square

The above lemma proves Theorem 3.4. The expression Δ is in E_{i+1} (and hence in F_{i+1}). Checking if it has an equivalent expression in F_i is undecidable as this can encode PCP.

B Details of Section 4

B.1 Upper Bounds

We first introduce some normal forms for expressions in E_i . Let U_i be the set of REWBs generated by the grammar $U_i ::= F_{i-1} \mid U_i \cdot U_i \mid a \downarrow_x (U_i)$. An expression in E_i is said to be in *Union Normal Form* (UNF) if it is of the form $u_i^1 + u_i^2 + \dots + u_i^r$, where u_i^j is an expression in U_i for every $j \in \{1, \dots, r\}$. From the semantics of REWBs, we infer that binding and concatenation distribute over union. By repeatedly applying this fact to any expression in E_i , we get the following result.

Proposition B.1. *For every expression e_i^1 in E_i , there exists a language equivalent one $u_i^1 + u_i^2 + \dots + u_i^r$ in UNF such that $|\mathcal{A}(u_i^j)| \leq |\mathcal{A}(e_i^1)|$ for every $j \in \{1, \dots, r\}$.*

Lemma 4.1. With an oracle for evaluating E_i queries, F_i queries can be evaluated in polynomial time.

Proof. Let G be the given data graph, f_i^1 be the query to be evaluated and ν be the given valuation for $fv(f_i^1)$. For every pair $\langle v_1, v_2 \rangle$ of nodes in G and every sub-expression e_i^1 of f_i^1 , check if $\langle v_1, v_2 \rangle \in e_i^1[\nu](G)$ by calling the oracle. Draw an edge labeled e_i^1 from v_1 to v_2 iff the oracle answers positively. Call the resulting data graph G' .

Perform the standard product construction of $\mathcal{A}(f_i^1)$ with G' (this can be done since G' also treats sub-expressions in E_i as a single letter). A pair $\langle v, v' \rangle$ belongs to $f_i^1[\nu](G)$ iff (v', q_f) is reachable from (v, q_0) in the product system, where q_f and q_0 are some final and initial states of $\mathcal{A}(f_i^1)$ respectively.

For the case of F_0 , the only sub-expressions that can not be handled directly by standard automata are those of the form $a[c]$. Given the evaluation ν , such expressions can be evaluated in linear time. Hence, in this case, the above procedure takes polynomial time without any oracle. \square

Theorem 4.2. For queries in E_i , the evaluation problem belongs to Σ_i .

Proof. By induction on i . For the base case $i = 1$, let e_1^1 be the given expression and let ν be the given valuation for the free variables of e_1^1 . We begin by non-deterministically choosing one of the sub-expressions for every sub-expression $e_1^2 + e_1^3$ of e_1^1 . This will result in an expression u_1^1 in U_1 . Now, to check if $\langle v, v' \rangle \in u_1^1[\nu](G)$, we proceed by recursion on the structure of u_1^1 as follows.

- To check if $\langle v_1, v_2 \rangle \in (u_1^2 \cdot u_1^3)[\nu](G)$, we non-deterministically guess a node v_3 and recursively check that $\langle v_1, v_3 \rangle \in u_1^2[\nu](G)$ and $\langle v_3, v_2 \rangle \in u_1^3[\nu](G)$.
- To check if $\langle v_1, v_2 \rangle \in a \downarrow_x (u_1^2)[\nu](G)$, we non-deterministically choose an a -successor v_3 of v_1 and note the data value d of the a -labeled edge from v_1 to v_3 . Next we recursively check that $\langle v_3, v_2 \rangle \in u_1^2[\nu[x \rightarrow d]](G)$.
- To check if $\langle v_1, v_2 \rangle \in f_0^1[\nu](G)$ for some expression f_0^1 in F_0 , we proceed as in the proof of Lemma 4.1.

Next we inductively assume that evaluating expressions in E_i is in Σ_i . To evaluate expressions in E_{i+1} , we proceed in the same way as in the base case. The only difference is in the case where we have to check $\langle v_1, v_2 \rangle \in f_i^1[\nu](G)$ for some expression f_i^1 in F_i . From Lemma 4.1, this can be done in polynomial time with an oracle for evaluating expressions in E_i . Since, by induction hypothesis, the oracle itself is in Σ_i , we conclude that evaluating expressions in E_{i+1} is in Σ_{i+1} . \square

Lemma B.2. *Suppose e_i^1 is an expression in E_i , with $|\mathcal{A}(e)| \leq k$ for every sub-expression e of e_i^1 . Let ν be a valuation for $\text{fv}(e_i^1)$. If there is a data path in $L(e_i^1, \nu)$ connecting v_1 to v_2 in a data graph with n nodes, then there is such a data path of length at most $(k^2n)^i$.*

Proof. By induction on i . For the base case $i = 1$, we begin by giving short witnesses for subexpressions of e_1^1 . For a subexpression f_0^1 , the automaton $\mathcal{A}(f_0^1)$ will have at most k states. Since the valuation does not change, we can infer from standard pumping arguments that if a data path in the language of f_0^1 connects v_1 to v_2 , there is such a data path of length at most kn . Next we consider e_1^1 . For every data path w in the language of e_1^1 , we infer from Proposition B.1 that there is an expression u_1^1 that contains w in its language. The path w may be split into sub-paths, each of which is in the language of some sub-expression of u_1^1 , of the form f_1^1 or $a \downarrow_x$. Since $|\mathcal{A}(u_1^1)| \leq k$, the number of such sub-expressions, and hence the number of sub-paths in w , is at most k . We have already seen that each sub-path can be replaced by one of length at most kn . Hence, the total length of the path is at most k^2n .

The induction step is similar, contributing a multiplicative factor of k^2n . Hence the result follows. \square

Theorem 4.3. Evaluating REWB queries in E_1 , parameterized by the size of the query is in W[P] .

Proof. We will use [3, Lemma 7, Theorem 8], which give machine characterizations for problems in W[P] . They prove that a parameterized problem is in W[P] iff there is a non-deterministic Turing machine that takes an instance (x, k) and decides the answer within $f(k)|x|^c$ steps, of which at most $f(k) \log |x|$ are non-deterministic (for some computable function f and a constant c). Such a Turing machine exists for evaluating REWB queries in E_1 . In such queries, every binding in the query is performed at most once in a path (since bindings are not iterable). Hence, the machine can first non-deterministically choose the data values for each binding in the query in the allowed number of non-deterministic steps. Then the expression can be treated as a standard regular expression, by substituting the guessed data values for the bindings. The set of data values found in the data graph can be considered as a finite alphabet and evaluation can be done in polynomial time using standard automata theoretic techniques. \square

Just like we get W[P] from W[SAT] by replacing formulas with circuits, we get AW[P] from AW[SAT] by replacing formulas with circuits. It has been proved in [3, Theorem 17] that a parameterized problem is in AW[P] iff there is an alternating Turing machine that takes an instance (x, k) and decides the answer within $f(k)|x|^c$ steps, of which at most $f(k) \log |x|$ are existential or universal (let us call such machines AW[P] machines). We have seen in Theorem 4.3 that evaluating REWB queries in E_1 can be done by non-deterministic Turing machines with bounded non-determinism (let us call them W[P] machines). As we did in Theorem 4.2, we can evaluate REWB queries in E_i using an oracle hierarchy of height i , consisting of W[P] machines. In complexity theory, an oracle hierarchy of NP machines is known to be equivalent to an alternating Turing machine. It is tempting to draw an analogous conclusion in parameterized complexity theory, saying that an oracle hierarchy of W[P] machines is equivalent to an AW[P] machine. However, we have not been able to prove such an equivalence for the following reason. In order to simulate oracle calls in an alternating machine, one generally needs as many non-deterministic steps as the number of calls to the oracle. In the oracle hierarchy of W[P] machines, the number of calls to an oracle may be polynomial in the size of the input, but the number of non-deterministic steps allowed in AW[P]

machines is logarithmic in the size of the input. We refer the interested reader to [3, Section 4] for some discussions on how some results in complexity theory fail in parameterized complexity theory.

For the query evaluation problem, we do not have upper bounds in parameterized alternating time bounded classes. However, we can get an upper bound in *uniform*-XNL, a parameterized space bounded class.

Theorem 4.4. Evaluating REWB queries, with size of the query as parameter, is in uniform-XNL.

Proof. We give a space bounded non-deterministic algorithm. Suppose n is the size of the data graph and k is the size of the expression and a pair of nodes is connected by a data path in the language of the expression. We know from Lemma B.2 that there is such a data path of length at most $((g(k))^2 n)^k$, where $g(k)$ is an upper bound on $|A(e)|$ for any REWB e of size k . A non-deterministic algorithm can guess and verify such a data path. It would have to store a counter to keep track of the length of the path, a valuation for variables in the expression and a node of the graph. All this needs space at most $\mathcal{O}((g(k))^2 \log n)$. \square

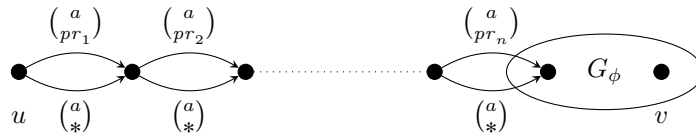
B.2 Lower Bounds

Lemma 4.5. Let ϕ be a Boolean formula over the propositional atoms pr_1, \dots, pr_n and $\nu : \{x_1, \dots, x_k\} \rightarrow \{pr_1, \dots, pr_n, *\}$ be a valuation. The source of G_ϕ is connected to its sink by a data path in $L(e_{eval}[k], \nu)$ iff ϕ is satisfied by the truth assignment that sets exactly the propositions in $\{pr_1, \dots, pr_n\} \cap \text{Range}(\nu)$ to true.

Proof. By induction on the structure of the Boolean formula. Suppose ϕ is a positively occurring atom pr_j , satisfied by the truth assignment. Hence, the data value pr_j is in $\{\nu(x_1), \dots, \nu(x_k)\}$. So the data path $\cdot \xrightarrow{\binom{a}{pr_1}} \cdot \xrightarrow{\binom{a}{pr_2}} \dots \xrightarrow{\binom{a}{pr_n}} \cdot \xrightarrow{\binom{pn?}{pr_j}} \cdot \xrightarrow{\binom{pa}{pr_j}} \cdot \xrightarrow{\binom{e}{*}} \cdot$ is in $L(a \downarrow_{x_{po}} a \downarrow_{x_{ne}} b \cdot pn?[x_{po}^-] \cdot pa[x_1^- \vee \dots \vee x_k^-]e, \nu)$, giving the desired data path in G_ϕ . Conversely, suppose that the source of G_ϕ is connected to its sink by a data path in $L(e_{eval}[k], \nu)$. Since $e_{eval}[k]$ begins with $a \downarrow_{x_{po}} a \downarrow_{x_{ne}}$ and G_ϕ begins with $\cdot \xrightarrow{\binom{a}{po}} \cdot \xrightarrow{\binom{a}{ne}} \cdot$, x_{po}, x_{ne} will have the values po, ne respectively. To reach the sink of G_ϕ , the data path $\cdot \xrightarrow{\binom{b}{*}} \cdot \xrightarrow{\binom{pn?}{po}} \cdot \xrightarrow{\binom{pa}{pr_j}} \cdot \xrightarrow{\binom{e}{*}} \cdot$ has to be in $L(b \cdot pn?[x_{po}^-] \cdot pa[x_1^- \vee \dots \vee x_k^-]e, \nu)$. This implies that the data value pr_j is in $\{\nu(x_1), \dots, \nu(x_k)\}$, which in turn implies that ϕ is satisfied by the truth assignment. The argument is similar for a negatively occurring propositional atom. The induction steps are standard arguments based on the semantics of Boolean formulas. \square

Theorem 4.6. For queries in E_1 , the evaluation problem is NP-hard.

Proof. We will reduce the satisfiability problem for Boolean formulas to the query evaluation problem. Suppose ϕ is a Boolean formula over the propositional atoms pr_1, \dots, pr_n . The data graph is as follows.



The ellipse at the end denotes the data graph G_ϕ corresponding to the Boolean formula ϕ , along with its source and sink nodes. The query to be evaluated on this is $a \downarrow_{x_1} a \downarrow_{x_2} \dots a \downarrow_{x_n} e_{eval}[n]$. To avoid too many parenthesis, we have not shown the scope of bindings. The scope of every binding extends till the end of the expression. We claim that the pair $\langle u, v \rangle$ is in the result of the query iff ϕ is satisfiable. Indeed, suppose $\langle u, v \rangle$ is in the result of the query. The data path from u to v will have two parts. The first one is in $L(a \downarrow_{x_1} a \downarrow_{x_2} \dots a \downarrow_{x_n})$ from u to the source of G_ϕ , resulting in a valuation $\nu : \{x_1, \dots, x_n\} \rightarrow \{pr_1, \dots, pr_n, *\}$. The second part is in $L(e_{eval}[n], \nu)$, connecting the source of G_ϕ to its sink. From Lemma 4.5, ϕ is satisfied by the truth assignment that sets pr_j to true iff $\nu(x_j) = pr_j$. Conversely, suppose ϕ is satisfied by some truth assignment $\alpha : \{pr_1, \dots, pr_n\} \rightarrow \{\text{true}, \text{false}\}$. Consider the data path from u to the source of G_ϕ that takes the edge labeled $\binom{a}{pr_j}$ if $\alpha(pr_j) = \text{true}$ and takes

the edge labeled $\binom{a}{*}$ otherwise. This path is in $L(a \downarrow_{x_1} a \downarrow_{x_2} \cdots a \downarrow_{x_n})$ and results in a valuation ν such that $\text{Range}(\nu) \cap \{pr_1, \dots, pr_n\}$ is precisely the set of propositional atoms set to true by the truth assignment α . Since this truth assignment satisfies ϕ , we conclude from Lemma 4.5 that the path can be continued from the source of G_ϕ to its sink. \square

The gadgets we present before Lemma 4.7 in the main paper build on earlier ideas and bring out the difference in the roles played by the data graph and the query, when reducing satisfiability to query evaluation. We begin with an observation about the REWB $e_{eval}[k]$.

Definition B.3 (Indistinguishable variables). *The variables x_1, \dots, x_k are said to be indistinguishable in an REWB e if they are free in e and for every condition c appearing in e , for every data value d and every pair of valuations ν and ν' with $\{\nu(x_1), \dots, \nu(x_k)\} = \{\nu'(x_1), \dots, \nu'(x_k)\}$, we have $d, \nu \models c$ iff $d, \nu' \models c$.*

The variables x_1, \dots, x_k are indistinguishable in $e_{eval}[k]$. The intuition is that $e_{eval}[k]$ treats the set $\{\nu(x_1), \dots, \nu(x_k)\}$ as the set of propositional atoms that are set to true. Any valuation ν' with $\{\nu'(x_1), \dots, \nu'(x_k)\} = \{\nu(x_1), \dots, \nu(x_k)\}$ will have the same meaning, as far as $e_{eval}[k]$ is concerned.

Suppose $PR = \{pr_1, \dots, pr_n\}$ is a set of propositional atoms and x_1, \dots, x_k are variables indistinguishable in some REWB e . We would like to check if the source of some graph G is connected to its sink by a data path in the language of e , for some injective valuation $\nu : \{x_1, \dots, x_k\} \rightarrow PR$. The data graph $G[\exists k/PR] \circ G$ and the expression $e[\exists k] \circ e$ defined in the main paper have been designed to achieve this.

Suppose ν is a valuation of some variables, whose domain does not intersect with $\{x_1, \dots, x_k\}$. We denote by $\nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$ the set of valuations ν' that extend ν such that $\text{domain}(\nu') = \text{domain}(\nu) \cup \{x_1, \dots, x_k\}$, ν' is injective on $\{x_1, \dots, x_k\}$ and $\{\nu'(x_1), \dots, \nu'(x_k)\} \subseteq PR$.

Lemma B.4. *Suppose x_1, \dots, x_k are indistinguishable in the REWB e and ν is a valuation for $fv(e) \setminus \{x_1, \dots, x_k\}$. The source of $G[\exists k/PR] \circ G$ is connected to its sink by a data path in $L(e[\exists k] \circ e, \nu)$ iff there exists $\nu' \in \nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$ and a data path in $L(e, \nu')$ connecting the source of G to its sink.*

Proof. Suppose the source of $G[\exists k/PR] \circ G$ is connected to its sink by a data path in $L(e[\exists k] \circ e, \nu)$. When this path reaches the source of G , the updated valuation ν' is in $\nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$. Hence, the continuation of the path from the source of G to its sink is in $L(e, \nu')$.

Conversely, suppose there exists $\nu' \in \nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$ and the source of G is connected to its sink by a data path in $L(e, \nu')$. There is a data path w_1 from the source of $G[\exists k/PR] \circ G$ to the source of G in $L(a_1^* a_1 \downarrow_{x_1} a_1^* a_1 \downarrow_{x_2} a_1^* \cdots a_1^* a_1 \downarrow_{x_k} a_1^*)$, resulting in a valuation $\nu'' \in \nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$ such that $\{\nu''(x_1), \dots, \nu''(x_k)\} = \{\nu'(x_1), \dots, \nu'(x_k)\}$. Since x_1, \dots, x_k are indistinguishable in e , we infer that the source of G is connected to its sink by a data path w_2 in $L(e, \nu'')$. The two data paths w_1 and w_2 can be concatenated to get a data path in $L(e[\exists k] \circ e, \nu)$ connecting the source of $G[\exists k/PR] \circ G$ to its sink. \square

Lemma 4.7. Let ϕ be a Boolean formula over the set PR of propositions and $k \in \mathcal{N}$. We can construct in polynomial time a data graph G and an REWB e_1^1 satisfying the following conditions.

1. The source of G is connected to its sink by a data path in $L(e_1^1)$ iff ϕ has a satisfying assignment of weight k .
2. The size of e_1^1 depends only on k .

Proof. The required data graph is $G[\exists k/PR] \circ G_\phi$ and e_1^1 is $e[\exists k] \circ e_{eval}[k]$. The correctness follows from Lemma B.4 and Lemma 4.5. \square

Lemma 4.9. Let $i \in \{1, \dots, k\}$ and ν_i be a valuation for $fv(e^i) \setminus \{x_1, \dots, x_i\}$. The source of G_i is connected to its sink by a data path in $L(e^i, \nu_i)$ iff for every $\nu \in \nu_i[\{x_1, \dots, x_i\} \rightarrow PR]$, there is a data path in $L(e^0, \nu)$ connecting the source of G_0 to its sink.

Proof. By induction on i . For the base case, we have $e^1 = b_1(a_1 \downarrow_{x_1} (e^0 a_1[x_1^-])c_1)^*$. Suppose there is a data path $w \in L(e^1, \nu_1)$ connecting the source of G_1 to its sink. We see from Figure 1 that w has to start from the edge labeled b_1 , followed by $\binom{a_1}{pr_1}$, assigning pr_1 to x_1 . Then w has to go from the source of G_0 to its sink using a sub-path in $L(e^0, \nu_1[x_1 \rightarrow pr_1])$. From the sink of G_0 , w is forced to take the edge labeled $\binom{a_1}{pr_1}$, in order to satisfy the condition $[x_1^-]$. The next letter of w is c_1 , which leads to a node from where the only outgoing edge is labeled with $\binom{a_1}{pr_2}$. This forces w to have a sub-path from the source to the sink of G_0 in $L(e^0, \nu_1[x_1 \rightarrow pr_2])$. We can similarly infer that for every $j \in \{1, \dots, n\}$, w has sub-paths in $L(e^0, \nu_1[x_1 \rightarrow pr_j])$ connecting the source of G_0 to its sink.

Conversely, suppose that for every $j \in \{1, \dots, n\}$, there is a data path $w_j \in L(e^0, \nu_1[x_1 \rightarrow pr_j])$ connecting the source of G_0 to its sink. The data path $b_1(\binom{a_1}{pr_j} w_j \binom{a_1}{pr_j} c_1)_{j \in \{1, \dots, n\}} \in L(e^1, \nu_1)$ connects the source of G_1 to its sink.

The induction step is similar to the base case. \square

Lemma 4.10. Let ν be a valuation for $fv(e) \setminus \{x_1, \dots, x_k\}$ for some REWB e . The source of $G[\forall k/PR] \circ G$ is connected to its sink by a data path in $L(e[\forall k] \circ e, \nu)$ iff for all $\nu' \in \nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$, the source of G is connected to its sink by a data path in $L(e, \nu')$.

Proof. Suppose the source of $G[\forall k/PR] \circ G$ is connected to its sink by a data path in $L(e[\forall k] \circ e, \nu)$. We infer from Lemma 4.9 that for all $\nu' \in \nu[\{x_1, \dots, x_k\} \rightarrow PR]$, there is a data path $w_{\nu'}$ from the source of G_0 to its sink in $L(e^0, \nu')$. For any such ν' that is injective on $\{x_1, \dots, x_k\}$, $w_{\nu'}$ can not have *skip* edges. The reason is that e^0 enforces $\nu'(x_i) = \nu'(x_j)$ for some distinct $i, j \in \{1, \dots, k\}$ in order to take a *skip* edge, but this is not possible for ν' if it is injective on $\{x_1, \dots, x_k\}$. Hence, for extensions ν' that are injective on $\{x_1, \dots, x_k\}$, the source of G is connected to its sink by a data path in $L(e, \nu')$.

Conversely, suppose that for all $\nu' \in \nu[\{x_1, \dots, x_k\} \xrightarrow{1:1} PR]$, the source of G is connected to its sink by a data path in $L(e, \nu')$. This data path also connects the source of G_0 to its sink, and is in $L(e^0, \nu')$. For valuations $\nu' \in \nu[\{x_1, \dots, x_k\} \rightarrow PR]$ that are not injective on $\{x_1, \dots, x_k\}$, there is a data path in $L(e^0, \nu')$ connecting the source of G_0 to its sink, which uses one of the *skip* edges from the source of G_0 to its sink. Hence, for all $\nu' \in \nu[\{x_1, \dots, x_k\} \rightarrow PR]$, there is a data path in $L(e^0, \nu')$ connecting the source of G_0 to its sink. We conclude from Lemma 4.9 that the source of $G[\forall k/PR] \circ G$ is connected to its sink by a data path in $L(e[\forall k] \circ e, \nu)$. \square

Lemma 4.11. Given an instance of the weighted quantified satisfiability problem, We can construct in polynomial time a data graph G and an REWB $e_{1+k_2+k_4+\dots}^1$ satisfying the following conditions.

1. The source of G is connected to its sink by a data path in $L(e_{1+k_2+k_4+\dots}^1)$ iff the given instance of the weighted quantified satisfiability problem is a yes instance.
2. The size of $e_{1+k_2+k_4+\dots}^1$ depends only on k_1, \dots, k_ℓ .

Proof. The required data graph G is $G[\exists k_1/PR_1] \circ G[\forall k_2/PR_2] \circ \dots \circ G_\phi$ and the required REWB $e_{1+k_2+k_4+\dots}^1$ is $e[\exists k_1] \circ e[\forall k_2] \circ \dots \circ e_{eval}[k_1 + \dots + k_\ell]$. We assume that \circ associates to the right, so $G_1 \circ G_2 \circ G_3$ is $G_1 \circ (G_2 \circ G_3)$ and $e^1 \circ e^2 \circ e^3$ is $e^1 \circ (e^2 \circ e^3)$. Suppose that the source of G is connected to its sink by a data path in $L(e_{1+k_2+k_4+\dots}^1)$. Lemma B.4 ensures that there exists an injective valuation $\nu_1 : \{x_1, \dots, x_{k_1}\} \rightarrow PR_1$ such that the source of $G[\forall k_2/PR_2] \circ \dots \circ G_\phi$ is connected to its sink by a data path in $L(e[\forall k_2] \circ \dots \circ e_{eval}[k_1 + \dots + k_\ell], \nu_1)$. Then Lemma 4.10 ensures that for all $\nu_2 \in \nu_1[\{x_{k_1+1}, \dots, x_{k_1+k_2}\} \xrightarrow{1:1} PR_2]$, the source of $G[\exists k_3/PR_3] \circ \dots \circ G_\phi$ is connected to its sink by a data path in $L(e[\exists k_3] \circ \dots \circ e_{eval}[k_1 + \dots + k_\ell], \nu_2)$. This argument can be repeated ℓ times to cover all alternations in the Boolean quantifiers. Finally, Lemma 4.5 ensures that the truth assignments corresponding to the valuations all satisfy the Boolean formula ϕ . The converse direction is similar. \square